PARAMETERS TUNING OF EVOLUTIONARY ALGORITHM FOR THE ORIENTEERING PROBLEM

Krzysztof Ostrowski

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: Various classes of algorithms solving optimization problems have some set of parameters. Setting them to appropriate values can be as important to results quality as choosing right algorithm components. Parameter calibration can be a complex optimization problem itself and many meta-algorithms were proposed to deal with it in a more automatic way. This paper presents automatic parameter tuning of an evolutionary algorithm solving the Orienteering Problem. ParamsILS method was chosen as a tuner. Obtained results show the importance of appropriate parameter setting in evolutionary algorithms: tuned algorithm achieved very high-quality solutions on known Orienteering Problem benchmarks.

Keywords: parameter tuning, evolutionary algorithms, Orienteering Problem

1. Introduction

The name Orienteering Problem (OP) derives from sport game of orienteering. Competitors start from a given point and have to reach another point within a prescribed time frame. In the meantime they can visit other control points and collect scores. The winner is the competitor who finishes with the highest score. The OP is an NP-hard optimization problem [4] and belongs to the family of Travelling Salesman Problems with Profits (TSPP) [26]. Its alternative name is Selective Travelling Salesman Problem (STSP) [15]. The OP has many practical applications including tourist trip planning, logistics and others. Various approaches (both exact and approximate) were proposed to solve the OP but because of its computational hardness most published methods were approximate. Evolutionary algorithms (EA) are among best known metaheuristics for solving optimization problems. When it comes to results quality EA parameters setting can be as important as choosing recombination operators. The paper presents parameter tuning (with ParamsILS method [30]) of an evolutionary algorithm which solves the Orienteering Problem. The article is organized as follows.

Advances in Computer Science Research, vol. 12, pp. 53-78, 2015.

Section 2 presents mathematical definition of the OP. Section 3 presents a review of the literature on the OP. Overview of parameter tuning methods is presented in section 4. In Section 5 the structure of the proposed evolutionary algorithm (EA) is described. Section 6 contains description of EA parameter tuning and experimental results. Conclusions are drawn and further work is suggested in Section 7.

2. Mathematical definition of the Orienteering Problem

Given a weighted graph (each edge has a non-negative cost and each vertex has some non-negative profit) the purpose of the Orienteering Problem (OP) is to find a simple path (or cycle) between a given pair of vertices (s - start vertex, e - end vertex) that maximizes total collected profit (sum of profits of visited vertices). The total cost of the path (sum of costs of visited edges) is limited by a given constraint (T_{max}). Let w_{ij} be a cost associated with edge (i, j) and p_i be a profit associated with vertex i. Let x_{ij} be a binary variable equal to 1 if a solution contains edge (i, j) and 0 otherwise. Let y_i be a binary variable equal to 1 if a solution contains vertex i and 0 otherwise. Let r_i be a position of vertex i in a solution - it is defined only for vertices included in a path. The OP can be formulated mathematically:

$$max \sum_{i \in V} \sum_{j \in V} (p_i \cdot x_{ij}) \tag{1}$$

$$\sum_{i \in V} \sum_{j \in V} w_{ij} \cdot x_{ij} \leq T_{max} \tag{2}$$

$$\sum_{j \in V} x_{sj} = \sum_{i \in V} x_{ie} = 1 \tag{3}$$

$$\bigvee_{k \in V \setminus \{s,e\}} (\sum_{i \in V} x_{ik} = \sum_{j \in V} x_{kj} \leq 1) \tag{4}$$

$$r_s = 1 \tag{5}$$

$$\bigvee_{i \in V} (y_i = 1 \Rightarrow 1 <= r_i <= n) \tag{6}$$

$$\bigvee_{i \in V,\ j \in V \setminus \{s\},} (x_{ij} = 1 \Rightarrow r_j = r_i + 1) \tag{7}$$

Maximization of total profit is associated with formula 1 and constraint 2 relates to maximal path cost which cannot exceed T_{max}. Constraint 3 indicates that the path

starts in vertex s and ends in vertex e while formula 4 implies that all other vertices can occur at most once on the path. Remaining constraints guarantee that the solution is a continuous path without subcycles.

3. Literature review of the Orienteering Problem

The Orienteering Problem was introduced by Tsiligirides [3]. Because of its NP-hardness exact solutions for the OP are only feasible in short time for graphs with a small number of nodes. The exact algorithms applied for the OP are the branch-and-cut and branch-and-bound methods. Fischetti et al. [14] have provided an exact solution tested for graphs with up to 500 vertices, and Gendreau et al. [12] have used the branch-and-cut method to exactly solve examples with up to 300 vertices. Ramesh et al. [8] have extended the branch-and-bound method via Lagrangian relaxation and have used it to solve examples with up to 150 control points. Exact algorithms for the OP usually need large amount of time to solve medium or large instances. Therefore, for practical applications, many researchers propose approximate methods to tackle the OP, based on a variety of approaches.

Tsiligirides [3] presents his S-algorithm for the OP, based on the Monte Carlo method. The algorithm constructs a large number of routes and chooses the one with the maximum profit. Then a deterministic heuristic algorithm partitions the geographic area into concentric circles and restricts the routes allowed to the sectors defined by the circles.

Golden et al. [4] introduced a three-step iterative heuristic involving route construction using a greedy algorithm and a centre-of-gravity heuristic. Ramesh et al. [7] propose a four-phase heuristic. After the best solution is chosen from iterations over a set of three phases (node insertion, edge exchange and node deletion), a fourth phase is entered, in which an attempt is made to insert unvisited nodes into the tour.

Wang et al. [27] use a neural network approach to solve the OP. They derive an energy function and a learning algorithm for a modified, continuous Hopfield neural network. Chao et al. [11] introduced a two-step iterative heuristic consisting of initialization and improvement steps. Tasgetiren [16] worked out the first genetic algorithm for the OP. The algorithm included tournament selection, injection crossover and mutation using elements of the local search method (add, omit, replace and swap operators). He also proposed a function that penalized infeasible solutions.

Gendreau et al. [13] present a tabu search heuristic for the OP. The algorithm iteratively inserts clusters of nodes into the current tour or removes a chain of nodes. Compared to the previous approaches, this method reduces the likelihood of getting trapped in a local optimum. Tests performed by the authors on randomly generated

instances with up to 300 nodes show that the algorithm yields very high-quality solutions.

Vansteenwegen et al. [29] developed the guided local search method (GLS) for the Team Orienteering Problem (TOP), which modifies other methods to reduce the likelihood of becoming trapped in a local optimum. The GLS meta-heuristic method yields satisfactory results for small-sized networks and is used in the Mobile Tourist Guide [32]. Solutions for the TOP generate m routes as results and total collected profit is maximized. Each node can be included at most one time in m resultant routes in the TOP. For $m = 1$ the GLS solves classic Orienteering Problem.

Schilde et al. [28] published two metaheuristics: Variable Neighbour Search (VNS) and Ant Colony Optimization (ACO). In relatively short execution time both algorithms achieved on benchmarks better average results than Chao's method [11] and GLS [29].

Souffriau et al. [31] applied Greedy Randomized Adaptive Search Procedure (GRASP) to solve the TOP. Campos et al. [35] successfully applied a path relinking (PR) method to GRASP to solve the OP. In the GRASP algorithm they propose an initial path containing only a starting and an ending vertex. Next, based on the ratio between greediness and randomness (four methods are possible), vertices are inserted one by one for as long as T_{max} is not exceeded. In the next step, the local search procedure (exchange and insert phase) attempts to reduce the length of the path and increase its total profit. In GRASPwPR a set of different solutions is created with the GRASP method, and path relinking is performed for each pair of solutions P and Q: the P path is gradually transformed into the Q by exchanging elements between P and Q. GRASPwPR results obtained on benchmark instances are one of the best among approximate solutions.

Author's Orienteering Problem research concentrated mainly on EA-based OP solutions for networks larger than known OP benchmarks (up to 908 vertices - network of cities in Poland) [33][34][36]. Obtained results showed advantage of evolutionary algorithms over known meta-heuristics (GLS, GRASP, GRASPwPR) for larger OP instances. In addition, experiments were also performed on Chao and Tsiligirides benchmark sets [39] and EA results (compared GLS and Chao methods) were promising. Author's purpose is to develop very effective metaheuristic not only for the OP but also for the Time-Dependent Orienteering Problem [38].

4. Overview of parameter tuning methods

Setting algorithm parameters *adhoc* can be troublesome and sometimes can result in poor quality of algorithm solutions. Various methods were proposed to deal with

this issue in a more automatic way. The simplest tuners are based on sampling of multi-dimensional parameter space. After testing the algorithm on a chosen set of parameter vectors the best vector is selected. Each vector evaluation is associated with running the algorithm on one or more test instances. Simplest way of choosing parameter vectors is full factorial design: we choose a few characteristic values for each parameter (for example quartiles of values ranges) and evaluate all possible combinations of these values. If there are n parameters and number of characteristic values is k for each parameter then number of evaluations is k^n. Therefore, in some cases full factorial experiment is too time consuming. Other sampling methods were proposed like Latin Hypercube Sampling and Taguchi Orthogonal Arrays [10]. In the first method parameter ranges are divided into a given number of m intervals. Afterwards m parameter vectors are randomly chosen with one constraint: for a given parameter no pair of vectors can have values from the same interval. The second method also enables to reduce vector number and is similar to full factorial design. Randomly generated vectors should meet the following condition: for each pair of parameters all combinations of parameters values are used the same number of times. It enables to reduce number of sampled vectors compared to full factorial design.

Simple sampling methods are often used as starting procedures to more complex tuning algorithms. For example instead of using one sampling procedure a tuning method can repeat it multiply and use previous sampling results to narrow parameter search space to a more promising area. Examples of such methods are Empirical Modelling of Genetic Algorithms [17] and CALIBRA [23].

Another approaches use models to estimate results for some parameter vectors instead of time-consuming algorithm tests. After some tests a model can be constructed. Its construction base on algorithm results obtained from a sample of parameter vectors. Afterwards, results from other parameter vectors can be estimated. Probably the most popular is regression model [20]. Instead of using only this one-stage procedure it is usually better to compose it with further actions. For example Coy et al. [18] use model-based approach, which is followed by a local search procedure. First, full factorial design over the whole parameter space is used to construct a linear regression model and to determine the path of steepest descent. Afterwards, this path is followed by a local search procedure and new vectors are generated and tested until the best solution stops changing. Sequential Parameter Optimization (SPO) [19] is a multi-stage procedure which updates the model after each iteration. After initial model construction the most promising vectors are tested and the results are used to update the model. This process is repeated until a given, maximum number of tests is reached. Unlike Coy's method this procedure improves the model after each itera-

tion.

Screening methods' purpose is to determine the best parameter vector from a given set by using as few tests as possible. They concentrate only on vectors that are most promising based on tests conducted so far. For example racing technique [21] bases on partial results obtained from tests conducted so far and at each step it discards parameter vectors that are not optimal according to statistical tests. Another approach (Iterative F-RACE) [24] repeats procedures of racing and model construction (multi-variate normal distribution) until a given number of tests is performed.

Finding parameter vectors that enable the algorithm to obtain high-quality results can be a difficult task. The objective function (algorithm results quality) in multi-dimensional parameter space is non-linear and can have multiple local optimums. In such situations approaches such as evolutionary algorithms (EA) or local search metaheuristics can be a good choice. In meta-EA a population consists of parameter vectors and the goal is to find an individual with highest meta-fitness function (solution quality of the tuned algorithm run with a given parameters set). Each evaluation in meta-EA is associated with running the tuned algorithm and therefore it can be very time consuming. One of the most advanced meta evolutionary algorithm is Relevance Estimation and Value Calibration of parameters (REVAC) [25]. It is a specific type of meta-EA where the population approximates the probability density function of the most promising areas of the parameter space. Furthermore, REVAC has been extended with Racing and Sharpening techniques in order to deal with the stochasticity of the utility values more effectively.

Parameters Iterated Local Search (ParamsILS) [30] is an example of local search approach to parameter tuning. After initial random sampling the best vector is chosen and a hill climbing procedure (first improvement) is applied to this vector until no better vector is found. Neighbourhood of a given vector v consists of vectors that differ from v only by one parameter value. After finding initial local optimum ParamILS repeatedly performs perturbation (random changes of some parameters) and hill climbing procedure. Perturbation enables to escape local optimum and to potentially search other optima. FocusedILS is a variant of ParamsILS that additionally uses racing when comparing parameter vectors.

5. Description of the proposed evolutionary algorithm

Proposed method for solving the Orienteering Problem is evolutionary algorithm with embedded local search operators. Path representation is used in the EA - each gene in a chromosome is equivalent to a path vertex. Path profit is treated as fitness value but infeasible solutions (paths longer than T_{max}) have zero fitness and are not allowed

in the population. At first, an initial population of P_{size} random routes is created. Afterwards, evolutionary phase takes place - operators of selection, crossover, mutation and disturb are applied repeatedly. Mutation, crossover and disturb operators can be either random or greedy (local search embedded) and ratio between randomness and greediness can be adjusted with parameters. Evolutionary phase terminates after a fixed number N_g of generations, or earlier if there have been no improvements in the last C_g generations. After the evolutionary phase all paths in the population undergo final local improvement procedure. The best feasible path (with highest total profit) obtained during algorithm run is EA final result. Three different crossover operators and three different selection methods (overall nine different algorithm configurations) were implemented and tested.

5.1 Initialization

The algorithm starts by generating an initial population of P_{size} routes. Each route is encoded as a sequence of different vertices. Let s and e be the given starting and ending points of the routes. Initialization process inserts new, random vertices into the route (at its end but before e vertex) as long as its total cost does not exceed T_{max} constraint. Only non-included vertices that can be inserted without violating T_{max} are considered during selection. Construction of the route ends when no new vertex can be inserted without exceeding T_{max}. The result of the initialization process are P_{size} random routes not exceeding a given limit T_{max}.

5.2 Selection

Three different selection procedures were tested.

Unbiased tournament selection: This method was presented by Sokolov et al. [22]. P_{size} tournaments are applied and in each tournament two random individuals are selected (tournament size is 2). The winners of each of P_{size} tournaments (individuals with the highest fitness) form parents pool. To reduce selection noise the standard procedure was modified: a random permutation p without fixed points is generated and tournaments are formed by pairs $(i, p(i))$. Thanks to this modification every individual takes part in exactly two tournaments. During crossover phase (after selection) children replace parents and create new generation.

Fitness proportionate selection with stochastic universal sampling: During this procedure P_{size} individuals are selected and form parents pool. Each selection is based

on fitness values and the probability of selecting individual i (fitness $f(i)$) is $\frac{f(i)}{F}$ where F is total fitness of whole population. In order to reduce selection bias standard roulette wheel was replaced by stochastic universal sampling [5]. Instead of P_{size} random samplings only one random value is used and individuals are selected from evenly spaced intervals. During crossover phase (after selection) children replace parents and create new generation.

Deterministic crowding: No parent selection is performed and selecting mechanism is applied after crossover procedures (survival selection). In deterministic crowding [9] each offspring competes with one of its parents and the fitter individual is chosen to next generation. To preserve population diversity competition is performed between more similar child-parent pairs. To measure differences between individuals edit distance is applied: it is computed by longest common subsequence (LCS) algorithm. The distance formula for paths i and j is

$$d(i,j) = L(i) + L(j) - 2 \cdot LCS(i,j) \tag{8}$$

where $L(i)$ is number of vertices in path i while $LCS(i,j)$ is the length of longest common subsequence of paths i and j. In order to speed up the algorithm initially size of vertex sets intersection is used instead of LCS in distance formula. If calculations suggests childA-parentB and childB-parentA competition pairs (instead of childA-parentA and childB-parentB pairs) then additional check is performed using LCS.

5.3 Crossover

First $\frac{P_{size} \cdot p_k}{2}$ different pairs of parents are randomly selected from the population (p_k - crossover probability) and each pair undergoes crossover procedure. Three crossover operators were tested:

2-point crossover: Crossover method applied by the author for the OP in [39] for the first time. At the beginning an ordered set S of common vertices for both parents is determined. Vertices order in the set is the same as in first parent. Next, chromosome fragments between two successive vertices in S are exchanged and two offspring are created. If any duplicates appear in offspring individuals outside of exchanged fragments they are immediately removed. There are two types of crossover: random (classic) version chooses crossover points randomly while greedy version maximizes fitness function of the fitter offspring. The ratio between greediness and randomness is determined by parameter z_k - it is equal to the probability of selecting

greedy crossover operator. Therefore z_k=0 means purely random crossover procedure while z_k=1 means purely greedy procedure. In fig. 1 there is an example of 2-point crossover.

Injection crossover: This method was applied by Tasgetiren in his OP-solving genetic algorithm [16]. A random insertion point is chosen in one parent and a random fragment in the other parent. The chosen fragment is inserted into the first parent, duplicates are removed from it and its size is reduced to previous size by cutting some genes. In this way an offspring individual is created. Tasgetiren applied random crossover method while in this paper greedy version of this operator was also created: insertion point is chosen to maximize offspring fitness. Ratio between randomness and greediness is determined by parameter z_k in the same way as in 2-point crossover.

Path relinking crossover: Path relinking method was used by Campos et al [35] as procedure complementing GRASP algorithm. It was also applied in genetic algorithm solving Orienteering Problem with Time Windows (OPTW) by Karbowska et al. [37]. In this method one route is gradually transformed into the other by inserting and removing vertices. The best intermediate route is chosen in greedy version of crossover. In random crossover version randomly selected intermediate route becomes an offspring. Ratio between randomness and greediness is determined by parameter z_k in the same way as in previous crossover operators.

One crossover method (2-point) creates two offspring individuals while remaining methods create only one offspring. To compensate this difference injection and path relinking crossover are doubled (performed two times on ordered parent pairs: P_1-P_2 and P_2-P_1).

5.4 Mutation

First $P_{size} \cdot p_m$ individuals are randomly selected from the population (p_m - mutation probability). Each of them undergoes mutation. First 2-opt procedure is performed once - it tries to change two edges in the path to shorten the path as much as possible (without changing vertex set). Example of edge exchanges is illustrated in fig. 2. From all such possible edge exchanges 2-opt chooses the option which shortens the path most.

After 2-opt procedure one vertex insertion or one vertex deletion is carried out. Probabilities of both insertion and deletion are the same (0.5). Both procedures have

parents

parent A: **1** - 3 - 9 - **7** - 4 - **5** - 12 - 11 - 14 - **16**

parent B: **1** - 6 - 8 - 2 - **7** - 13 - 10 - **5** - 15 - **16**

option I

offspring 1: **1** - 6 - 8 - 2 - **7** - 4 - **5** - 12 - 11 - 14 - **16**

offspring 2: **1** - 3 - 9 - **7** - 13 - 10 - **5** - 15 - **16**

option II

offspring 1: **1** - 3 - 9 - **7** - 13 - 10 - **5** - 12 - 11 - 14 - **16**

offspring 2: **1** - 6 - 8 - 2 - **7** - 4 - **5** - 15 - **16**

option III

offspring 1: **1** - 3 - 9 - **7** - 4 - **5** - 15 - **16**

offspring 2: **1** - 6 - 8 - 2 - **7** - 13 - 10 - **5** - 12 - 11 - 14 - **16**

Fig. 1. Example of 2-point crossover. Given two parents which have four common points (in bold: 1, 7, 5, 16) there are three possible fragment exchanges between successive points. Option I shows children created by fragments exchange between vertices 1 and 7, option II illustrates children created by segments exchange between points 7 and 5 while in option III one can see offspring individuals made by exchange of fragments between vertices 5 and 16. Random variant of crossover chooses random option while greedy variant chooses the option with highest fitness value of the better child.

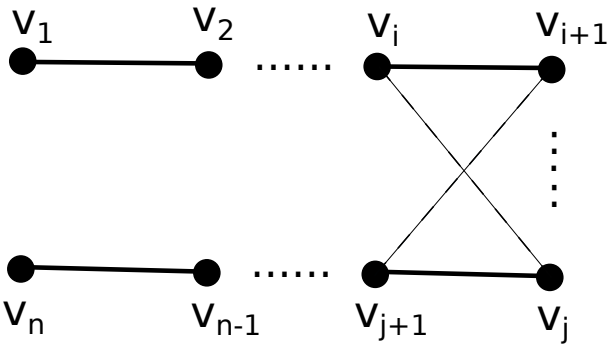

Fig. 2. Example of 2-opt edge change. For a path $(v_1, v_2, ..., v_i, v_{i+1}, ..., v_j, v_{j+1}, ..., v_n)$ two edges $((v_i, v_{i+1})$ and $(v_j, v_{j+1}))$ are replaced by two other edges $((v_i, v_j)$ and (v_{i+1}, v_{j+1}) - green) and this exchange results in a path $(v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_{j+1}, ..., v_n)$.

two types: greedy (local search) and random. Greedy versions of insert and delete methods chose most favourable vertices in terms of their profits and change of path cost. During greedy insert all non-present vertices and all insertion points are considered and the option with highest ratio $\frac{ProfitIncrease}{CostIncrease}$ is chosen. Similarly local search used in delete procedure removes the vertex which minimizes ratio $\frac{ProfitDecrease}{CostDecrease}$. Random versions choose randomly selected vertices for insertion/deletion. However, random insert procedure adds new, random vertex in the insertion point which minimizes route cost increase. The ratio between greediness and randomness is determined by parameter z_m in the same way as by parameter z_k in crossover phase. Mutation examples are shown in fig. 3 and 4. Mutation operators do not allow paths to exceed T_{max} constraint. In order to speed up local search procedure (in insertion mutation) for a given number of vertex pairs (i, j) ordered lists of vertices are stored - they are sorted by $\frac{ProfitIncrease}{CostIncrease}$ heuristic associated with inserting a given vertex between vertices i and j.

5.5 Disturb

Disturb procedure is a different type of mutation and causes bigger changes in routes. First $P_{size} \cdot p_z$ individuals are randomly selected from the population (p_z - disturb probability). Each of them undergoes disturb procedure which removes some route fragment (but no more than 10 percent of route vertices). Again there are two types of disturb procedure: greedy variant from all fragments of a given length chooses the one which minimizes ratio $\frac{ProfitDecrease}{CostDecrease}$ while random variant chooses route segment randomly. The ratio between greediness and randomness is determined by parameter

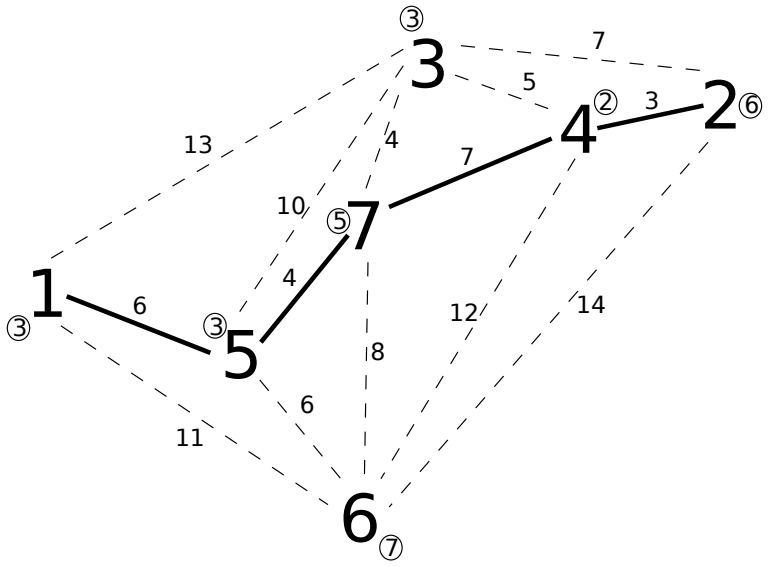

Fig. 3. Example of inserting mutation. Given a graph with 7 vertices and a path (1, 5, 7, 4, 2) there are two vertices not present in the path (3 and 6). Each of them can be inserted into one of 4 insertion points. That gives a total of 8 insertion options. Greedy insertion chooses the option maximizing $\frac{ProfitIncrease}{CostIncrease}$ ratio - inserting vertex 3 between vertices 7 and 4 gives the best ratio of 1.5 (profit increase is 3 and path cost increase is 2). Random insertion chooses random vertex and inserts it into a place which minimizes path cost increase (either vertex 3 between vertices 7 and 4 or vertex 6 between vertices 5 and 7. Graph weights are given beside edges and profits beside vertices (in circles).

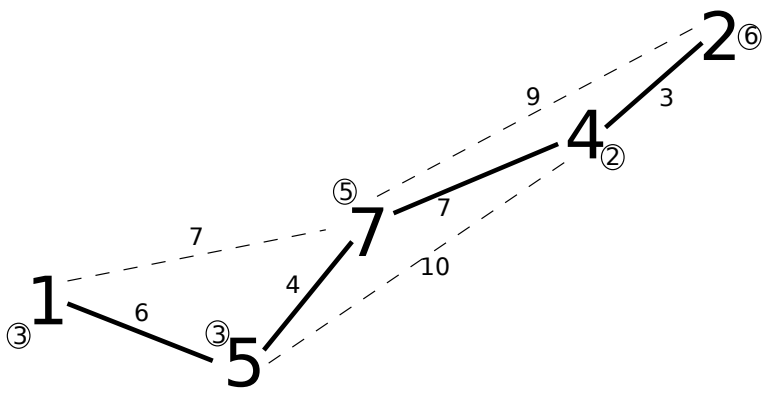

Fig. 4. Example of deleting mutation. Given a path (1, 5, 7, 4, 2) there are three options of vertex deletion (5, 7 or 4). Greedy deletion chooses the option minimizing $\frac{ProfitDecrease}{CostDecrease}$ ratio - removing vertex 5 gives the best ratio of 1.0 (profit decrease is 3 and path cost reduction is 3). Random deletion chooses random vertex from all candidates. Graph weights are given beside edges and profits beside vertices (in circles).

z_z. This procedure enables routes to escape from local optimum and gives a chance to explore a different fragment of solution space.

5.6 Local improvement phase

After evolutionary phase is over each path undergoes final, local improvement procedure. First a series of 2-opt procedures is performed on a given path. Afterwards one greedy vertex deletion and a series of greedy vertex insertions are performed until T_{max} is reached (insertions and deletions described in mutation section). This process is repeated until no improvement is found. This procedure also tries to improve the best path found by the algorithm during evolutionary phase by performing a series of greedy insertions until T_{max} is reached. In practice, for fine-tuned EAs this phase usually gives only small improvement.

6. Parameter tuning of EA

After choosing algorithm components parameter tuning was carried out. This process was performed for each of 9 EA configurations (3 different selections x 3 different crossovers).

6.1 Tuning methodology

Params Iterated Local Search (ParamsILS) [30] algorithm was used for parameter tuning because of its simplicity and lower time consumption compared to meta-EAs. This meta-algorithm searches multi-dimensional parameter space using local search procedures. Parameter vectors (meta-solutions) processed by ParamsILS are evaluated by averaging results from multiple runs of the tuned EA. In the first phase N vectors are randomly chosen and the best of meta-solutions is selected for hill-climbing procedure. In this local improvement phase neighbourhood of the current solution is searched in a random order and the current solution is updated if a better neighbour is found. Hill-climbing procedure is performed as long as current solution has any better neighbour. Afterwards disturb procedure occurs: random S parameters of the best found meta-solution are changed (escaping from local optimum) and hill-climbing is performed again. Procedures of disturb and hill-climbing are iterated until a specific number of evaluations (or amount of time) is reached (during this experiment time limit for tuning was set to 4 hours). In some iterations current solution is again randomly initialized (large jump in solution space) - the probability of random restart is $P_{restart}$. Neighbourhood of parameter vector v is defined as all vectors that differs from v only on one parameter. In this case parameter discretization is

needed. In table 1 there are descriptions of all tuned parameters and their set of values used during tuning procedure. Therefore meta-algorithm operates on 6-dimensional parameter space. Disturb procedure of the tuned EA has additional, small values of probability because it modifies large parts of routes and can be destructive to solutions quality when overused. Some basic EA parameters were set to specific values (table 2). Population size is a compromise between exploration ability and computation time. Parameters associated with generations number were determined during earlier experiments and should not stop EA prematurely.

Table 1. Tuned parameters of EA

parameter	**description**	**values set**
p_k	crossover probability	{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
p_m	mutation probability	{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
p_z	disturb probability	{0, 0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1}
z_k	crossover greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}
z_m	mutation greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}
z_z	disturb greediness/randomness ratio	{0, 0.2, 0.4, 0.6, 0.8, 1}

Table 2. Set parameters of EA

parameter	**description**	**value**
P_{size}	population size	100
Ng	maximum number of generations	5000
Cg	maximum number of generations without improvement	500

In order to better explore meta-solution space first phase of the algorithm was modified. For each parameter p (with values range $<p_{min}, p_{max}>$) two subranges of values were defined: lower subrange $<p_{min}, \frac{p_{min}+p_{max}}{2})$ and upper subrange $<\frac{p_{min}+p_{max}}{2}, p_{max}>$. If number of parameters is n then parameter space can be divided into 2^n parts. Two vectors belong to the same part only if all of their parameters belong to the same subrange. In the first part of the meta-algorithm 2^6 random vectors from different parts of parameter space are selected. After initial selection meta-algorithm works in the way described above. Parameters of tuner are presented in table 3.

Table 3. Parameters of meta-algorithm

parameter	**description**	**value**
N	number of initial random vectors	64
S	number of parameters changed during disturb phase	2
$P_{restart}$	probability of random restart of current solution	0.3

Table 4. All problem instances with T_{max} values

Class	Instance name	T_{max}	Class	Instance name	T_{max}	
I	kroA100	10641	II	eil101A	158	
	kroB100	11071		cmt121A	137	
	kroC100	10375		cmt151A	175	
	kroD100	10647		cmt200A	191	
	kroE100	11034		gil262A	595	
	rd100	3955		eil101B	315	
	eil101	315		cmt121B	273	
	lin105	7190		cmt151B	350	
	pr107	22152		cmt200B	382	
	gr120	3471		gil262B	1189	
	pr124	29515		eil101C	472	
	bier127	59141		cmt121C	409	
	pr136	48386		cmt151C	525	
	gr137	34927		cmt200C	573	
	pr144	29269		gil262C	1784	
	kroA150	13262				
	kroB150	13065				
	pr152	36841				
	u159	21040				
	rat195	1162				
	d198	7890				
	kroA200	14684				
	kroB200	14719				
	ts225	63322				
	pr226	40185				
	gil262	1189				
	pr264	24568				
	pr299	24096				
	lin318	21015				
	rd400	7641				

6.2 Problem instances

All OP instances (divided into two classes) are presented in table 4. Class I tests are Travelling Salesman Problem (TSP) instances adapted to OP by Fischetti et al. [14]. They come from TSPLIB library (XML format of distance matrices) created by Reinelt [6] and profits of vertices were generated by Fischetti according to pseudo-random formula:

$$p_i = 1 + (7141 \cdot i + 73)(mod\, 100) \tag{9}$$

where p_i is profit of vertex i. T_{max} values for class I instances were set as 50 percent of shortest hamiltonian cycles. Distances between vertices were truncated to integer numbers.

Class II tests were Vehicle Routing Problem (VRP) instances adapted to OP by Fischetti et al [14]. Some of them (eil101, gil262) come from TSPLIB library while other instances were created by Christofides et al. [2]. In these instances customer demands from VRP were interpreted as vertices profits in the OP. T_{max} values were set as 25, 50 and 75 percent of shortest hamiltonian cycles. Depending on T_{max} value instances have additional letter (A, B or C) in their names. For each instance from both classes number of vertices is encoded into its name. Distances between vertices were rounded to nearest integers. For all instances of both classes paths start and end in vertex 1.

6.3 Tuning results

All experiments (calibration and testing) were carried out on Intel i7 3.6 GHz processor. Programs were implemented in C++ and executed in Linux operating system. Calibration process was carried out on three problem instances: pr299 and rd400 (from class I) as well as gil262 (from class II). From all OP instances with known exact solutions these three were hardest to obtain close to optimal solutions for published methods and for evolutionary algorithm during earlier tests. Evaluation of a given parameter vector consisted of 30 EA runs (10 runs for each of the calibration networks) and the result was average gap to the optimal solution. The gap was calculated as $100 \cdot (1 - \frac{P_{alg}}{P_{opt}})$ where P_{alg} is route profit obtained by EA while P_{opt} is profit of optimal route. Tuning process was performed for nine different EA configurations (various types of selection and crossover).

In table 5 calibration results are presented for nine different EA configurations. It can be seen that generally the hardest instance (in terms of obtained results) is rd400 - large number of vertices (400) is one of reasons for biggest gaps among three calibration networks. In terms of average gap the best results were obtained when

Table 5. Calibration results for different EA configurations with best found sets of parameters and average gaps to optimal results (their 95 percent confidence intervals given beside them) for calibration networks. Crossover types: 2P - two point crossover, INJ - injection crossover, PR - path relinking crossover. Selection types: TUR - unbiased tournament selection, SUS - fitness proportionate selection with stochastic universal sampling, CRO - deterministic crowding. The result of the best EA configuration in bold.

Crossover	Selection	Calibrated parameters		rd400	pr299	gil262C	All 3 networks
		p_k p_m p_z	z_k z_m z_z	gap (%)	gap (%)	gap (%)	avg. gap (%)
2-P	TUR	0,6 1,0 0,70	0,4 0,6 0,4	3,47	3,32	1,14	2,64 ±0.50
	SUS	0,6 1,0 0,10	0,8 0,8 0,6	2,37	1,42	0,81	1,53 ±0.38
	CRO	1,0 1,0 0,10	0,6 0,8 1.0	0,63	0,91	0,38	**0.64 ±0.12**
INJ	TUR	0,8 1,0 0,50	0,6 0,6 0,6	4,84	2,40	1,53	2,92 ±0.56
	SUS	0,6 1,0 0,00	0,8 0,8 -	3,20	2,37	0,74	2,10 ±0.54
	CRO	1,0 1,0 0,00	0,4 0,6 -	5,96	2,34	2,54	3,61 ±0.32
PR	TUR	0,6 1,0 0,70	1,0 0,6 0,6	3,86	1,90	0,63	2,13 ±0.60
	SUS	0,4 1,0 0,10	1,0 0,8 0,6	2,09	2,63	0,41	1,71 ±0.34
	CRO	0,8 1,0 0,03	0,4 0,8 0,2	2,36	0,93	0,45	1,25 ±0.30

Table 6. Comparison of average gap (in percent) for rd400 instance (average from 30 runs) depending on z_k and z_m greediness/randomness parameters (for crossover and mutation respectively). EA configuration: 2-point crossover and deterministic crowding. Remaining parameters were the same as in table 5. 95 percent confidence intervals for average gap are given by ± sign. Average gaps after evolution phase but before final local improvement phase are given in parentheses. The best parameter set in bold.

$z_k \setminus z_m$	0.0	0.2	0.4	0.6	0.8	1.0
0.0	11.07 ±0.45 (29.40)	8.06 ±0.46 (18.67)	4.51 ±0.32 (7.81)	2.11 ±0.30 (2.90)	2.28 ±0.23 (2.79)	2.49 ±0.21 (2.93)
0.2	8.44 ±0.69 (13.64)	5.04 ±0.28 (6.58)	2.54 ±0.21 (2.77)	0.79 ±0.17 (0.83)	1.10 ±0.15 (1.13)	1.03 ±0.16 (1.07)
0.4	6.97 ±0.68 (8.97)	4.23 ±0.27 (4.65)	2.34 ±0.16 (2.42)	0.79 ±0.14 (0.83)	0.76 ±0.14 (0.78)	0.89 ±0.15 (0.93)
0.6	6.59 ±0.59 (7.33)	3.79 ±0.21 (4.04)	2.31 ±0.19 (2.34)	1.01 ±0.11 (1.04)	**0.69** **±0.10** **(0.71)**	0.91 ±0.14 (0.93)
0.8	6.14 ±0.55 (6.42)	3.83 ±0.22 (3.98)	2.38 ±0.14 (2.42)	1.19 ±0.13 (1.20)	0.79 ±0.09 (0.80)	0.84 ±0.09 (0.87)
1.0	6.19 ±0.38 (6.36)	3.89 ±0.17 (3.91)	2.62 ±0.12 (2.66)	1.42 ±0.11 (1.43)	0.96 ±0.12 (0.97)	1.00 ±0.12 (1.02)

calibrating EA with two-point crossover type and deterministic crowding survival selection type. Average gap is only 0.64 per cent and EA gives the best results for all three calibration networks. One can see that algorithm configurations with two-point crossover give the best average results regardless of selection type. Tuned EAs with deterministic crowding selection give the best results in two (out of three) crossover types. Crossover probabilities chosen by tuner were generally high or medium and best configurations had very high crossover probability (0.8-1.0). Crossover greediness/randomness ratio is medium or high and its values for best configurations (0.4-0.6) suggest importance of both elements (greediness and randomness) when creating offspring individuals. Mutation probability is very high (1.0) in all cases and its greedy-random ratio is medium or high (similarly to crossover). These results show importance of local search operations in mutation phase. It can be seen that disturb probability varies strongly depending on selection type - it is high for tournament selection and low for other selection types. It is associated with stronger tournament selection pressure compared to other selection types - increasing disturb probability counters selection effect and assures better diversity (and longer convergence) of the population.

In table 6 there are results (rd400 test instance) for the best EA configuration (2-point crossover and deterministic crowding) depending on z_k and z_m values. Remaining parameters were set by tuning procedure. It can be seen that solution quality rises quickly as greediness of crossover and mutation grows. The difference between purely random parameters configuration and the best configurations is about 10 percent. One can see that gaps for randomness-favouring parameter sets are even bigger without post-evolution local improvement phase. For more greedy parameter sets final improvement phase brings little or no change to solutions quality. The best parameters values are about 0.6-0.8 for z_m and 0.4-0.8 for z_k. When greediness grows to 1.0 (for both parameters) results quality drops gently. One can see that variance of EA results (shown by length of confidence intervals) is dependent on z_k and z_m ratios - more greedy configurations give steadier results. Confidence interval for gap in the best parameter set is rather short but a few other parameter sets are not significantly worse in terms of results. Local search methods are essential in this configuration but random component also plays its role. Large advantage of greediness over randomness arises from selection mechanism - in deterministic crowding selection pressure is lower and randomness-favouring parameter sets cannot converge to good solutions. In this algorithm configuration local search operators are probably more responsible for convergence than selection.

Table 7. Experiment results for different, tuned EA configurations (average gaps and 95 percent confidence intervals for them) for all test instances from both problem classes. Crossover types: 2P - two point crossover, INJ - injection crossover, PR - path relinking crossover. Selection types: TUR - unbiased tournament selection, SUS - fitness proportionate selection with stochastic universal sampling, CRO - deterministic crowding. The results of the best EA configuration in bold.

Crossover	Selection	calibration networks	class I networks	class II networks
		avg. gap (%)	avg. gap (%)	avg. gap (%)
2-P	TUR	2.64	2.12 ±0.10	2.06 ±0.16
	SUS	1.53	1.20 ±0.08	1.21 ±0.10
	CRO	**0.64**	**0.41 ±0.03**	**0.34 ±0.03**
INJ	TUR	2.92	2.66 ±0.13	2.90 ±0.23
	SUS	2.10	2.01 ±0.12	1.53 ±0.18
	CRO	3,61	1.52 ±0.04	2.24 ±0.08
PR	TUR	2.13	2.06 ±0.13	1.71 ±0.11
	SUS	1.71	1.51 ±0.10	1.30 ±0.12
	CRO	1.25	1.55 ±0.09	1.43 ±0.12

6.4 Test instances results

For each problem instance EA was executed 30 times and average profit was computed. In addition 95 percent confidence intervals for average gaps were presented. In table 7 average results of tuned EA configurations for all instances from both classes are displayed and compared to results from calibration phase. One can see that the best algorithm configuration from tuning phase (2-point crossover and deterministic crowding survival selection) achieves on average the best results on all test instances as well - average gap is only 0.41 and 0.34 percent (for class I and class II respectively). In most cases better results for calibration networks implied superiority for all instances but exceptions also appeared (configuration injection crossover + deterministic crowding behaves much better overall than for tuning networks). It can be seen that in most cases overall average gaps for class I and class II are smaller than average gaps for calibrated networks (despite fitting EAs to them). It results from the fact that calibration was performed on larger instances (and probably the hardest to obtain high-quality results) with 262-400 vertices while most test networks had 100-200 nodes.

Table 8. Comparison of results of the best tuned EA configuration (2-point crossover + deterministic crowding) with results of GRASP, GRASPwPR and best known solutions (class I). Execution time is given in seconds. 95 percent confidence intervals for gaps are given beside gaps.

Instance	EA				GRASP		GRASP PR		Best solution
	profit	gap (%)		time	profit	gap (%)	profit	gap (%)	
kroA100	3177.8	0.10	±0.05	1.3	3135	1.45	3181	0	3181
kroB100	3191	0.13	±0.00	1.3	3183	0.38	3191	0.13	3195
kroC100	3025.7	0.60	±0.32	1.5	3044	0	3044	0	3044
kroD100	3222.3	0.11	±0.02	1.7	3152	2.29	3212	0.43	3226
kroE100	3303.9	0.18	±0.17	1.2	3260	1.51	3310	0	3310
rd100	3448.7	0.61	±0.07	1.1	3449	0.61	3453	0.49	3470
eil101	3667.4	0.02	±0.03	1.1	3596	1.96	3645	0.63	3668
lin105	3576.7	0.01	±0.01	1.4	3577	0	3577	0	3577
pr107	2681	0.00	±0.00	0.8	2681	0	2681	0	2681
gr120	4198.5	0.58	±0.13	1.4	4138	2.01	4201	0.52	4223
pr124	3840	0.00	±0.00	1.8	3840	0	3840	0	3840
bier127	5374.7	0.02	±0.03	4.1	5154	4.13	5254	2.27	5376
pr136	4214.2	0.21	±0.04	1.7	4170	1.26	4213	0.24	4223
gr137	4272.1	0.44	±0.03	1.7	4255	0.84	4284	0.16	4291
pr144	3911.5	2.07	±0.57	2	3902	2.3	3994	0	3994
kroA150	4916.8	0.04	±0.03	2.3	4768	3.07	4915	0.08	4919
kroB150	5014.5	0.05	±0.09	2.1	4967	1	5001	0.32	5017
pr152	4192.4	0.09	±0.05	1.9	4094	2.43	4175	0.5	4196
u159	5028.3	0.31	±0.08	2.3	4809	4.66	4987	1.13	5044
rat195	5895.1	0.69	±0.08	2.9	5693	4.09	5693	4.09	5936
d198	6507.9	0.48	±0.07	3.3	6347	2.94	6476	0.96	6539
kroA200	6583.3	0.49	±0.06	3.3	6447	2.55	6551	0.98	6616
kroB200	6581.6	0.23	±0.08	3.5	6357	3.64	6409	2.85	6597
ts225	6732.1	1.17	±0.32	4.2	6701	1.63	6784	0.41	6812
pr226	6685	0.09	±0.10	6.5	6375	4.72	6614	1.15	6691
gil262	9135.8	0.25	±0.05	5.6	8847	3.41	8941	2.38	9159
pr264	6666	0.00	±0.00	3.8	6666	0	6666	0	6666
pr299	9010	1.07	±0.14	6.3	8645	5.07	8689	4.59	9107
lin318	10795.6	1.52	±0.12	8.4	10074	8.1	10339	5.68	10962
rd400	13461.3	0.69	±0.10	16.2	12365	8.78	12365	8.78	13555
Avg.	**5410.4**	**0.41**	**±0.03**	**3.2**	**5256.4**	**2.49**	**5322.8**	**1.29**	**5437.2**

Table 9. Comparison of results of the best tuned EA configuration (2-point crossover + deterministic crowding) with results of GRASP, GRASPwPR and best known solutions (class II). Execution time is given in seconds. 95 percent confidence intervals are given beside gaps.

Instance	EA				GRASP		GRASPwPR		Best solution
	profit	gap (%)		time	profit	gap (%)	profit	gap (%)	
eil101A	571.9	0.02	±0.02	0.7	566	1.05	572	0	572
cmt121A	408.9	0.75	±0.27	0.7	412	0	412	0	412
cmt151A	824	0.00	±0.00	0.8	815	1.09	824	0	824
cmt200A	1204.7	0.02	±0.02	2	1145	4.98	1181	1.99	1205
gil262A	4492.9	0.38	±0.20	2.4	3916	13.17	4050	10.2	4510
eil101B	1047.1	0.18	±0.08	1.2	1024	2.38	1032	1.62	1049
cmt121B	712.7	0.32	±0.17	1.5	699	2.24	707	1.12	715
cmt151B	1535.7	0.08	±0.03	2.1	1482	3.58	1528	0.59	1537
cmt200B	2172.4	1.16	±0.15	5.2	2073	5.69	2105	4.23	2198
gil262B	8420.6	0.42	±0.09	5.7	7946	6.03	8074	4.52	8456
eil101C	1333.6	0.18	±0.06	2.7	1295	3.07	1302	2.54	1336
cmt121C	1129.5	0.40	±0.13	2.1	1120	1.23	1125	0.79	1134
cmt151C	1993.3	0.48	±0.04	6	1965	1.9	1996	0.35	2003
cmt200C	2873.3	0.27	±0.04	10.9	2791	3.12	2824	1.98	2881
gil262C	11153.6	0.37	±0.02	13.3	10938	2.3	11046	1.33	11195
Avg.	**2658.3**	**0.34**	**±0.03**	**3.8**	**2545.8**	**3.46**	**2585.2**	**2.08**	**2668.5**

Table 10. New, best solution found for gil262A.

instance	profit	route
gil262A	4510	1-164-225-83-158-250-63-238-178-70-191-124-119-4-216- 105-142-26-247 -209-181-6-4-217-47-188-162-180-129-49-197-175-144-99-241-118-93-179-211- 42-111-110-21-228-30-98-133-172-182-60-163-76-220-16-103-40-39-224- 44-226-242-12-215-132-58-204-102-115-149-27-10-154-231-171-3-126-65-1

In table 8 there is a comparison (problem class I) between best tuned EA configuration and other algorithms (GRASP, GRASPwPR) [35] as well as best known results obtained by branch-and-cut exact algorithm (results in [35] but algorithm proposed by [14]). EA with average gap of only 0.41 percent is almost 1 percent better than GRASPwPR and over 2 percent better than GRASP. Confidence intervals for average gap show that EA results are significantly better for most test instances. EA advantage is very clear for larger instances (3-8 percent over GRASPwPR). In most cases EA obtains close to optimal results in reasonably short execution time (less than 5 seconds in most cases). In table 9 there is a similar comparison for problem class II. EA again clearly outperforms GRASP and GRASPwPR methods by 3.1 and 1.7 percent respectively (gaps are statistically significant) and its execution time is still reasonably short. Results obtained by tuned EA are clearly better than other methods and on average are very close to the best known solutions obtained by exact algorithm. For one instance (gil262A) new, best solution (profit 4510) was obtained by EA (presented in table 10) - previous best solution (profit 4466) was obtained by branch-and-cut algorithm but for some networks its execution time limit (5 hours) was reached and resulting solutions could be worse than optimal.

7. Conclusions and further research

In the paper parameter tuning of evolutionary algorithm solving the Orienteering Problem was carried out. Nine different algorithm configurations (varying in selection and crossover phases) were tuned and tested. Parameters of EA were tuned with ParamsILS local search algorithm. Results show the importance of both choosing algorithm components and parameter calibration when developing EAs. Parameter tuning done in an automatic way have advantages over *adhoc* calibration. For a given algorithm configuration the described tuner was able to find a very good parameters set in a few hours. Tuned EA achieved high-quality solutions and clearly outperformed GRASP and GRASPwPR methods (reaching results close to optimal for most test instances).

Further research is concentrated on other aspects of OP solving evolutionary algorithms i.e. different types of paths initialization (local search methods), infeasible solutions in the population and combining a few crossover operators in one EA. The author is also working on solutions for the Time-Dependent Orienteering Problem (TDOP) [38] particularly for trip planners in public transport networks. Methods working well for the classic OP (EAs in particular) can probably be adapted successfully to time-dependent version of the problem.

Acknowledgment

The author gratefully acknowledges support from the Polish Ministry of Science and Higher Education at the Bialystok University of Technology (grant W/WI/4/2014).

References

[1] Croes, G. A.: A method for solving traveling salesman problems, Operations Research, vol. 6, 791-812, 1958.
[2] Christofides, N., Mingozzi, A., Toth, P.: The Vehicle Routing Problem. Combinatorial Optimization, 315-338, 1979.
[3] Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society, vol. 35 (9), 797-809, 1984.
[4] Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics, vol. 34, 307-318, 1987.
[5] Baker, J.E.: Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the Second International Conference on Genetic Algorithms and their Application, Hillsdale, New Jersey: L. Erlbaum Associates, 14-21, 1987.
[6] Reinelt, G.: TSPLIB - A Travelling Salesman Problem Library. ORSA Journal of Computing, vol. 3, 155-165, 1991.
[7] Ramesh, R., Brown, K.: An efficient four-phase heuristic for the generalized orienteering problem. Computers and Operations Research. vol 18, 151-165, 1991.
[8] Ramesh, R., Yoon, Y., Karwan, M.: An optimal algorithm for the orienteering tour problem. ORSA Journal on Computing, vol. 4, 155-165, 1992.
[9] Mahfoud, S.W.: Crowding and preselection revisited. Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN II), Brussels, Belgium, 1992. Elsevier, Amsterdam, The Netherlands, 27-36, 1992.
[10] Taguchi, G., Yokoyama, T.: Taguchi Methods: Design of Experiments, ASI Press, 1993.
[11] Chao, I., Golden, B., Wasil, E.: Theory and methodology - a fast and effective heuristic for the orienteering problem. European Journal of Operational Research, vol. 88, 475-489, 1996.
[12] Gendreau, M., Laporte, G., Semet, F.: A branch-and-cut algorithm for the undirected selective traveling salesman problem, Networks, vol. 32(4), 263-273, 1998.
[13] Gendreau, M., Laporte, G., Semet, F.: A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research, vol. 106, 539-545, 1998.

[14] Fischetti, M., Salazar, J., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing, vol. 10, 133-148, 1998.
[15] Feillet, D., Dejax, P., Gendreau, M.: Traveling Salesman Problems With Profits, An Overview. Transportation Science, vol. 38, 188-205, 2001.
[16] Tasgetiren, M.: A genetic algorithm with an adaptive penalty function for the orienteering problem. Journal of Economic and Social Research, vol. 4 (2), 1-26. 2001.
[17] Myers, R., Hancock, E.A.: Empirical modelling of genetic algorithms, Evolutionary Computation, vol. 9, 461-493, 2001.
[18] Coy, S. P., Golden, B. L., Runger, G. C., Wasil, E. A.: Using experimental design to find effective parameter settings for heuristics, Journal of Heuristics, vol. 7, 77-97, 2001.
[19] Bartz-Beielstein, T., Parsopoulos, K., Vrahatis, M.: Analysis of Particle Swarm Optimization Using Computational Statistics, in Chalkis (Ed.), Proceedings of the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2004), Wiley, pp. 34-37, 2004.
[20] Ramos, I., Goldbarg, R., Goldbarg, E., Neto, A.: Logistic regression for parameter tuning on an evolutionary algorithm, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation, IEEE Press, Edinburgh, UK, vol. 2, 1061-1068, 2005.
[21] Birattari, M.: Tuning Metaheuristics, Springer, 2005.
[22] Sokolov, A., Whitley, D.: Unbiased tournament selection, Proceedings of Genetic and Evolutionary Computation Conference. ACM Press, 1131-1138, 2005.
[23] Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search, Oper. Res. , vol. 54, 99-114, 2006.
[24] Balaprakash, P., Birattari, M., Stutzle, T.: Improvement strategies for the F-Race algorithm: Sampling design and iterative refinementace algorithm: Sampling design and iterative refinement, in: T. Bartz-Beielstein, M. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 4771, 108-122, 2007.
[25] Nannen, V., Eiben, A. E.: Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters, in: M. M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, 1034?1039, 2007.
[26] Jozefowiez, N., Glover, F., Laguna, M.: Multi-objective Meta-heuristics for the Traveling Salesman Problem with Profits. Journal of Mathematical Modelling and Algorithms, vol. 7, 177-195, 2008.

[27] Wang Q.,Sun X., Golden B. L. and Jia J.,: Using artificial neural networks to solve the orienteering problem, emph Annals of Operations Research, vol.61, 111-120, 2008.

[28] Schilde, M., Doerner, K., Hartl, R., Kiechle, G.: Metaheuristics for the biobjective orienteering problem. Swarm Intelligence, vol. 3, 179-201, 2009.

[29] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G. and Oudheusden, D.V.: A guided local search metaheuristic for the team orienteering problem. European Journal of the Operational Research, vol. 196(1), 118-127, 2009.

[30] Hutter, F., Hoos, H.H., Leyton-Brown, K., Stutzle, T.: ParamILS: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research, vol. 36, 267-306, 2009.

[31] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G. and Oudheusden, D.V.: A path relinking approach for the team orienteering problem, Computers and Operations Research, vol. 37, 1853-1859, 2010.

[32] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: The City Trip Planner: An expert system for tourists, Expert Systems with Applications, vol. 38(6), 6540-6546, 2011.

[33] Ostrowski, K, Koszelew, J.: The comparison of genetic algorithm which solve orienteering problem using complete and incomplete graph, Zeszyty Naukowe, Politechnika Bialostocka. Informatyka, vol. 8 61-77, 2011.

[34] Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K., Zabielski, P.: Genetic algorithm solving orienteering problem in large networks, Frontiers in Artificial Intelligence and Applications, vol. 243, 28-38, 2012.

[35] Campos, V., Marti, R.,Sanchez-Oro, J., Duarte, A.: Grasp with Path Relinking for the Orienteering Problem. Journal of the Operational Research Society, vol. 156, 1-14, 2013.

[36] Koszelew. J., Ostrowski. K.: A Genetic Algorithm with Multiple Mutation which Solves Orienteering Problem in Large Networks. Computational Collective Intelligence. Technologies and Applications, LNCS 8083, 356-366, 2013.

[37] Karbowska-Chilinska, J., Zabielski, P.: Genetic Algorithm with Path Relinking for the Orienteering Problem with Time Windows, Fundamenta Informaticae, vol. 135, 419-431, 2014.

[38] Ostrowski, K.: Comparison of Different Graph Weights Representations Used to Solve the Time-Dependent Orienteering Problem, Trends in Contemporary Computer Science, Podlasie 2014, Bialystok University of Technology Publishing Office, 144-154, 2014.

[39] Zabielski. P., Karbowska-Chilinska, J., Koszelew, J., Ostrowski, K.: A Genetic Algorithm with Grouping Selection and Searching Operators for the Orienteering Problem, Lecture Notes in Artificial Intelligence, LNCS 9012, 31-40, 2015.

KALIBRACJA PARAMETRÓW ALGORYTMU EWOLUCYJNEGO ROZWIĄZUJĄCEGO ORIENTEERING PROBLEM

Streszczenie Różne klasy algorytmów rozwiązujących problemy optymalizacyjne posiadają zestawy parametrów. Ustawienie odpowiednich wartości parametrów może być równie ważne, co dobór odpowiednich komponentów algorytmu. Kalibracja parametrów sama w sobie może być skomplikowanym problemem optymalizacyjnym i wiele meta-algorytmów zostało zaproponowanych by przeprowadzać ten proces automatycznie. Artykuł prezentuje automatyczną kalibrację parametrów algorytmu ewolucyjnego rozwiązującego Orienteering Problem. W tym celu wybrano metodę ParamsILS. Otrzymane rezultaty ukazują jak ważny jest odpowiedni dobór parametrów: algorytm po kalibracji uzyskał bardzo wysokiej jakości rozwiązania dla znanych sieci testowych.

Słowa kluczowe: kalibracja parametrów, algorytmy ewolucyjne, Orienteering Problem